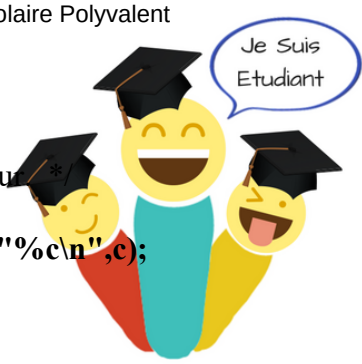


printf("BONJOUR"); /* pas de retour à la ligne du curseur apres l'affichage, */
printf("BONJOUR\n"); /* affichage du texte, puis retour à la ligne du curseur */
Affichage d'un caractère: La fonction **putchar** permet d'afficher un caractère:
c étant une variable de type **char**, l'écriture **putchar(c);** est équivalente à **printf("%c\n",c);**

Affichage d'un texte: La fonction **puts** permet d'afficher un texte:
l'écriture **puts("bonjour");** est équivalente à **printf("bonjour\n");**

Il vaut mieux utiliser puts et putchar si cela est possible, ces fonctions, non formatées, sont d'exécution plus rapide, et nécessitent moins de place en mémoire lors de leur chargement.

clrscr(); /* efface l'écran */

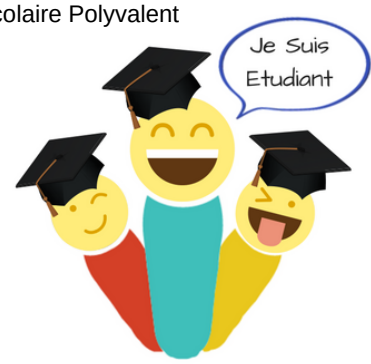
INCREMENTATION - DECREMENTATION

i = i+1; est équivalent à i++;
i = i-1; est équivalent à i--;

LES DECLARATIONS DE CONSTANTES

Exemple:
```
void main()
{
const float PI = 3.14159;
float perimetre,rayon = 8.7;
perimetre = 2*rayon*PI;
....
}
```
OU
```
#define PI = 3.14159;
void main()
{
float perimetre,rayon = 8.7;
perimetre = 2*rayon*PI;
....
}
```

```
#include <stdio.h>          /* bibliotheque d'entrees-sorties standard */
#include <conio.h>
void main()
{
puts("BONJOUR");/* utilisation d'une fonction-bibliotheque */
puts("Pour continuer frapper une touche...");
getch();  /* Attente d'une saisie clavier */
}
```

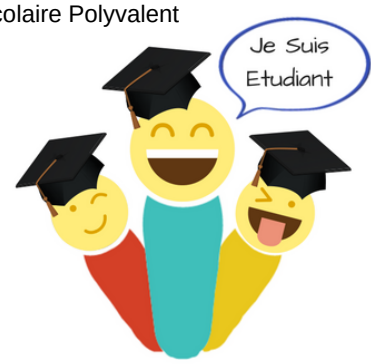



```
#include <stdio.h>          /* bibliotheque d'entrees-sorties standard */
#include <conio.h>
void main()
{
int a, b, somme ;  /* déclaration de 3 variables */
puts("BONJOUR");/* utilisation d'une fonction-bibliotheque */
a = 10 ; /* affectation* /
b = 50 ;  /* affectation */
somme = (a + b)*2 ;  /* affectation et opérateurs   */
printf(« Voici le resultat : %d\n », somme) ;
puts("Pour continuer frapper une touche...");
getch();  /* Attente d'une saisie clavier */
}
```

Le langage C distingue plusieurs types d'entiers:

TYPE	DESCRIPTION	TAILLE MEMOIRE
float	réel standard	4 octets
double	réel double précision	8 octets
int	entier standard signé	4 octets: $-2^{31} \leq n \leq 2^{31}-1$
unsigned int	entier positif	4 octets: $0 \leq n \leq 2^{32}$
short	entier court signé	2 octets: $-2^{15} \leq n \leq 2^{15}-1$
unsigned short	entier court non signé	2 octets: $0 \leq\leq n \leq 2^{16}$
char	caractère signé	1 octet : $-2^{7} \leq\leq n \leq 2^{7}-1$
unsigned char	caractère non signé	1 octet : $0 \leq n \leq 2^{8}$

CARACTERE		VALEUR (code ASCII)	NOM ASCII
'\n'	interligne	0x0a	LF
'\t'	tabulation horizontale	0x09	HT
'\v'	tabulation verticale	0x0b	VT
'\r'	retour charriot	0x0d	CR
'\f'	saut de page	0x0c	FF
'\\'	backslash	0x5c	\
'\"	cote	0x2c	'



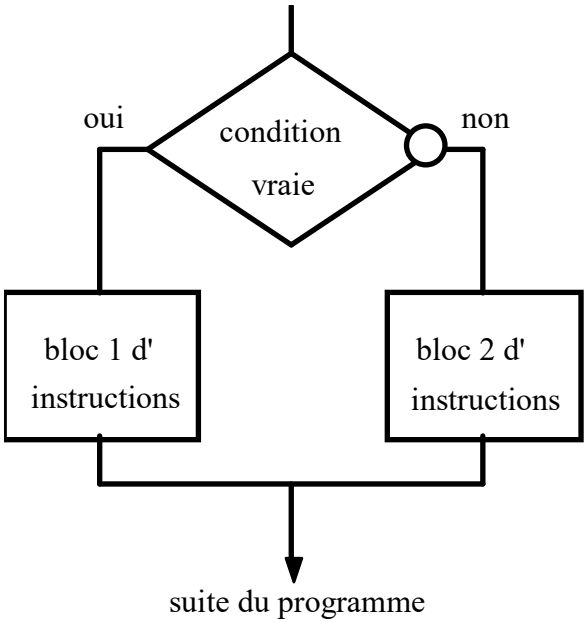
'\"'	guillemets	0x22	"

LES BOUCLES

L'INSTRUCTION SI ... ALORS ... SINON ...

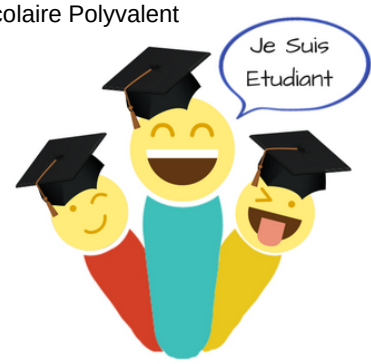
Il s'agit de l'instruction: **si (expression conditionnelle vraie)**
alors {BLOC 1 D'INSTRUCTIONS}
sinon {BLOC 2 D'INSTRUCTIONS}

Organigramme:

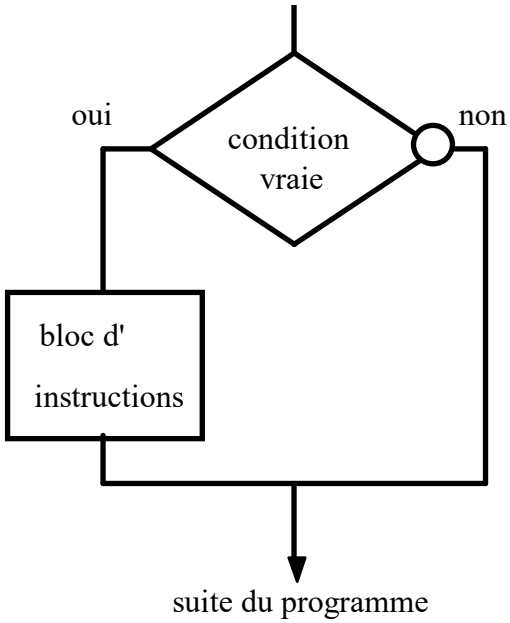


Syntaxe en C:

```
if (expression)
        {
        .....;          /* bloc 1 d'instructions */
        .....;
        .....;
        }
        else
        {
        .....;          /* bloc 2 d'instructions */
        .....;
        .....;
        }
```



Le bloc "sinon" est optionnel: **si (expression vraie)**
alors {BLOC D'INSTRUCTIONS}



Syntaxe en C:

```
if (expression)
        {
        .....;          /* bloc d'instructions */
        .....;
        .....;
        }
```

Remarque: les {} ne sont pas nécessaires lorsque les blocs ne comportent qu'une seule instruction.

LES OPERATEURS LOGIQUES

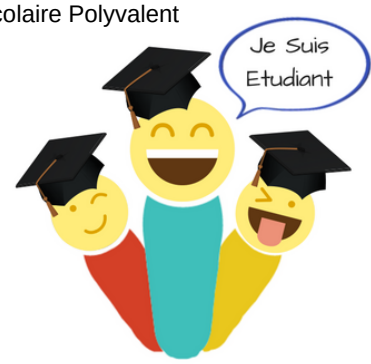
test d'égalité: **if (a==b)** **" si a égal b "**

test de non égalité: **if (a!=b)** **" si a différent de b "**

tests de relation d'ordre: **if (a<b)** **if (a<=b)** **if (a>b)** **if (a>=b)**

test de ET LOGIQUE: **if ((expression1) && (expression2))**
" si l'expression1 ET l'expression2 sont vraies "

test de OU LOGIQUE **if ((expression1) || (expression2))**
" si l'expression1 OU l'expression2 est vraie "



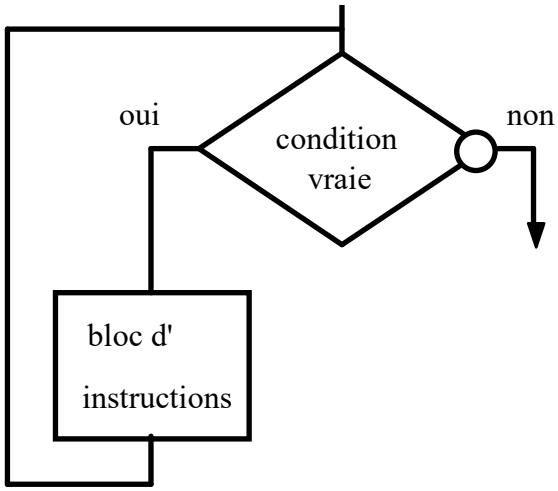
test de NON LOGIQUE **if (!(expression1))**
" si l'expression1 est fausse "

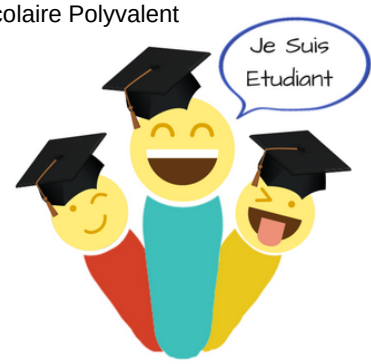
Toutes les combinaisons sont possibles entre ces tests.

LA BOUCLE TANT QUE ... FAIRE ...

Il s'agit de l'instruction: **tant que (expression vraie)**
faire{BLOC D'INSTRUCTIONS}

Organigramme:





Syntaxe en C: **while (expression)**
{
.....; **/* bloc d'instructions */**
.....;
.....;
}

Le test se fait **d'abord,** le bloc d'instructions n'est pas forcément exécuté.

Remarque: les {} ne sont pas nécessaires lorsque le bloc ne comporte qu'une seule instruction.

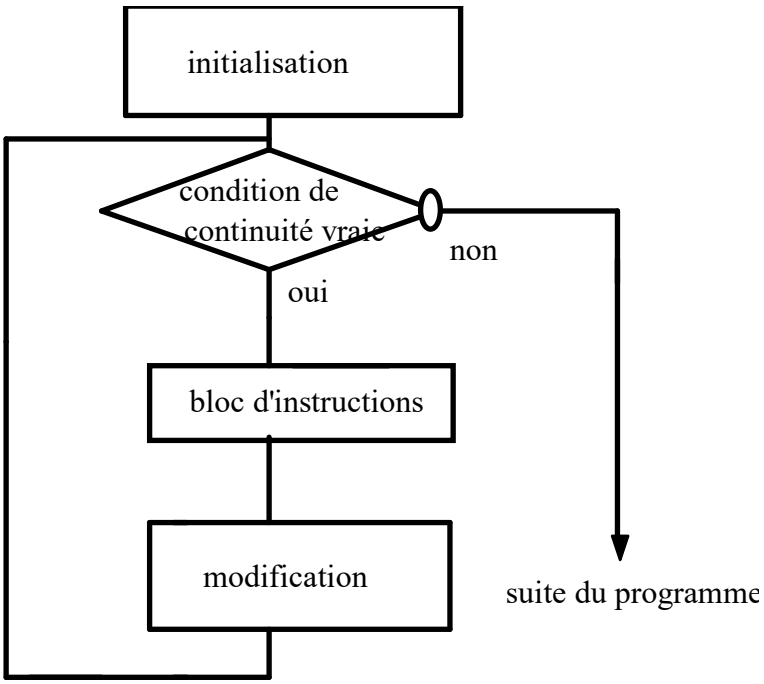
Remarque: On peut rencontrer la construction suivante: **while (expression);** terminée par un ; et sans la présence du bloc d'instructions. Cette construction signifie: **"tant que l'expression est vraie attendre".**

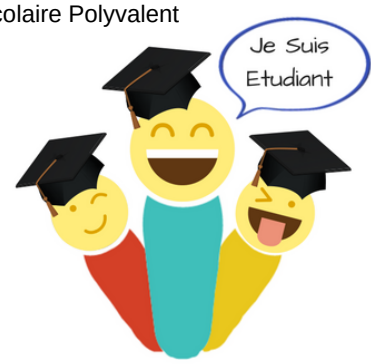
L'INSTRUCTION POUR ...

Il s'agit de l'instruction:

pour (initialisation; condition de continuité vraie;modification)
{BLOC D'INSTRUCTIONS}

Organigramme:





Syntaxe en C:

for(initialisation ; condition de continuité ; modification)
{
.....; /* bloc d'instructions */
.....;
.....;
}

Remarques:
Les {} ne sont pas nécessaires lorsque le bloc ne comporte qu'une seule instruction.
Les 3 instructions du for ne portent pas forcément sur la même variable.
Une instruction peut être omise, mais pas les ;

L'INSTRUCTION AU CAS OU ... FAIRE ...

L'instruction switch permet des choix multiples **uniquement sur des entiers (int) ou des caractères (char).**

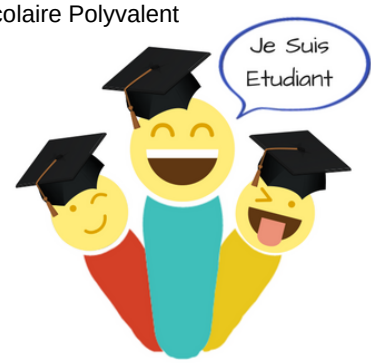
Syntaxe:

switch(variable de type char ou int) au cas où la variable vaut:
{
case valeur1: .....; - cette valeur1: executer ce bloc d'instructions.
.....;
break;
valeur2:.....; - cette valeur2: executer ce bloc d'instructions.
.....;
break;
.
. etc ...
.
default: .....; - aucune des valeurs précédentes: executer ce bloc
.....; d'instructions, pas de "break" ici.

}

le bloc "default" n'est pas obligatoire.

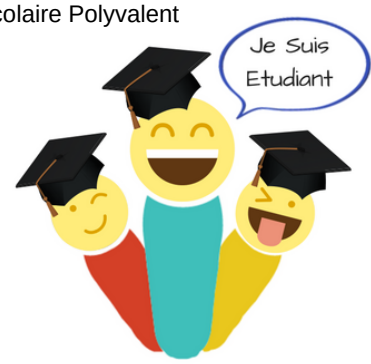
L'instruction switch correspond à une cascade d'instructions if ...else

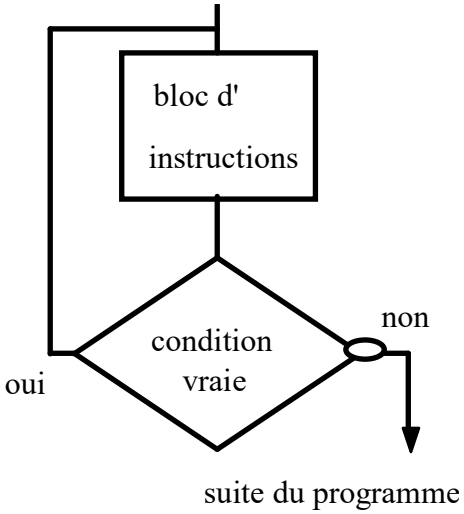


L'INSTRUCTION REPETER ... TANT QUE ...

Il s'agit de l'instruction: **répéter{BLOC D'INSTRUCTIONS}**
tant que (expression vraie)

Organigramme:





Syntaxe en C: **do**

```
{
.....;          /* bloc d'instructions */
.....;
.....;
}
while (expression);
```

Le test se faisant **après,** le bloc est exécuté au moins une fois.

Remarque: les {} ne sont pas nécessaires lorsque le bloc ne comporte qu'une seule instruction.

CHAPITRE 6

LES TABLEAUX ET LES CHAINES DE CARACTERES

LES TABLEAUX DE NOMBRES (INT ou FLOAT)

Les tableaux correspondent aux matrices en mathématiques. Un tableau est caractérisé par sa taille et par ses éléments.

Les tableaux à une dimension:

Déclaration: **type nom[dim];** Exemples: **int compteur[10];**
float nombre[20];

Cette déclaration signifie **que le compilateur réserve dim places en mémoire pour ranger les éléments du tableau.**

Exemples:
int compteur[10]; le compilateur réserve des places pour 10 entiers, soit 20 octets en TURBOC et 40 octets en C standard.
float nombre[20]; le compilateur réserve des places pour 20 réels, soit 80 octets.

Remarque: dim est nécessairement une VALEUR NUMERIQUE. Ce ne peut être en aucun cas une combinaison des variables du programme.

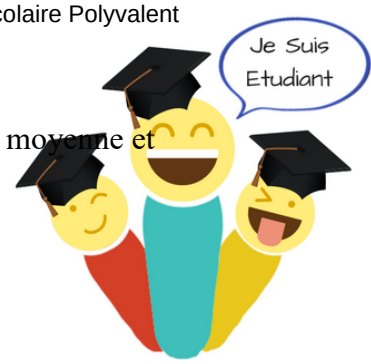
Utilisation: Un élément du tableau est repéré par son indice. En langage C les tableaux commencent à l'indice 0. L'indice maximum est donc dim-1.

Appel: **nom[indice]**

Exemples:
```
compteur[2] = 5;
nombre[i] = 6.789;
printf("%d",compteur[i]);
scanf("%f",&nombre[i]);
```

Tous les éléments d'un tableau (correspondant au dimensionnement maximum) ne sont pas forcément définis.

D'une façon générale, les tableaux consomment beaucoup de place en mémoire. On a donc intérêt à les dimensionner au plus juste.



Exercice VI_1: Saisir 10 réels, les ranger dans un tableau. Calculer et afficher la moyenne et l'écart-type.

Les tableaux à plusieurs dimensions:

Tableaux à deux dimensions:

Déclaration: **type nom[dim1][dim2];** Exemples: **int compteur[4][5];**
float nombre[2][10];

Utilisation: Un élément du tableau est repéré par ses indices. En langage C les tableaux commencent aux indices 0. Les indices maximum sont donc dim1-1, dim2-1.

Appel: **nom[indice1][indice2]**

Exemples:
```
compteur[2][4] = 5;
nombre[i][j] = 6.789;
printf("%d",compteur[i][j]);
scanf("%f",&nombre[i][j]);
```

Tous les éléments d'un tableau (correspondant au dimensionnement maximum) ne sont pas forcément définis.

Exercice VI_2: Saisir une matrice d'entiers 2x2, calculer et afficher son déterminant.

Tableaux à plus de deux dimensions:

On procède de la même façon en ajoutant les éléments de dimensionnement ou les indices nécessaires.

INITIALISATION DES TABLEAUX

On peut initialiser les tableaux au moment de leur déclaration:

Exemples:

```
int liste[10] = {1,2,4,8,16,32,64,128,256,528};

float nombre[4] = {2.67,5.98,-8,0.09};

int x[2][3] = {{1,5,7},{8,4,3}};  /* 2 lignes et 3 colonnes */
```